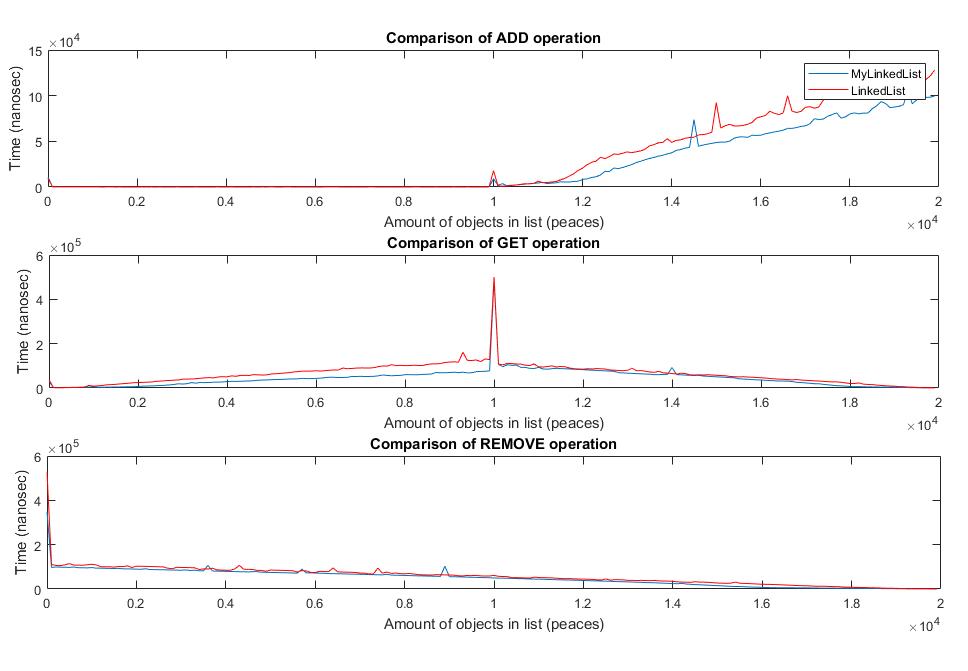
Отчет по домашней работе на тему «Java collections»

**Задание 1.** Сравнение производительности, написанной мной коллекции с java реализацией.

Сравнение делаю по трем основным операциям: добавление, поиск, удаление элемента. Количество элементов в коллекциях 20000. Время выполнения получаю для каждой из операций считаю для каждого элемента массива через System.nanoTime(), т.о. получаю 20000 моментов времени для операции ADD, 20000 моментов времени для операции GET, 20000 моментов времени для операции REMOVE. Построил графики зависимости времени выполнения от количества элементов в коллекции для каждой из операций.



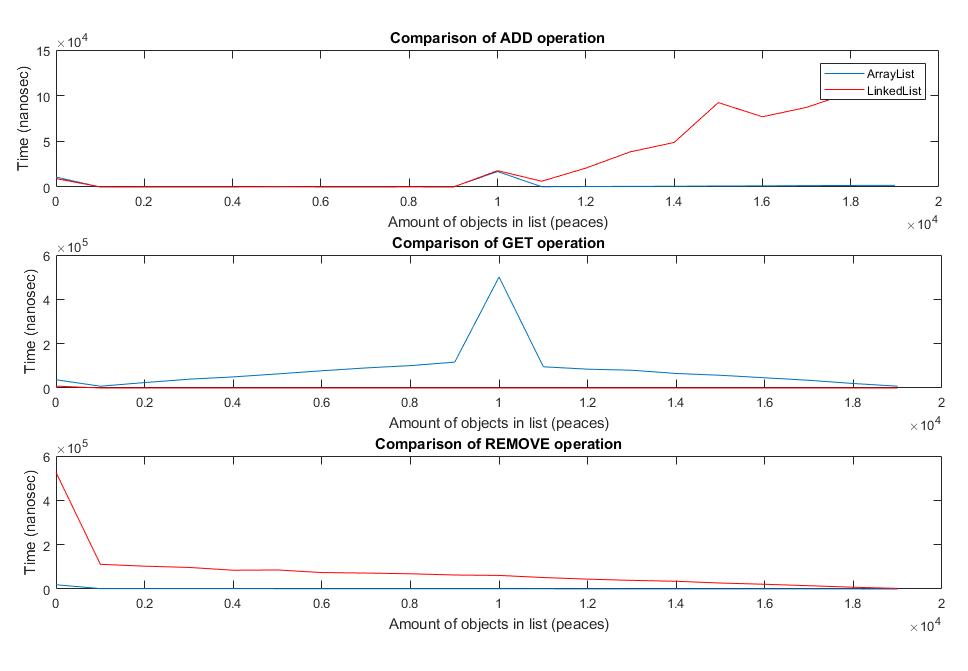
Красным цветом - Java LinkedList, синим - моя реализация LinkedList.

Операцию add выполняю следующим образом: первую половину элементов (10000 элементов) записываю последовательно в конец и получаю в итоге производительность 0(1). После этого начинаю записывать постоянно в индекс, равный половине размера коллекции (в моем случае это 10000). После этого получаю рост затраченного времени на запись. Связываю это с тем, что теперь для поиска нужного индекса операции add(index,element) приходится проходить всё больше элементов от конца списка.

Как таковых сильных отличий в операциях не нашёл. Судя по графикам мой MyLinkedList затрачивает меньше времени для вставки элемента в середину коллекции. По операциям get и remove существенных отличий нет.

**Задание 2.**

1. Сравнение производительности ArrayList и LinkedList.



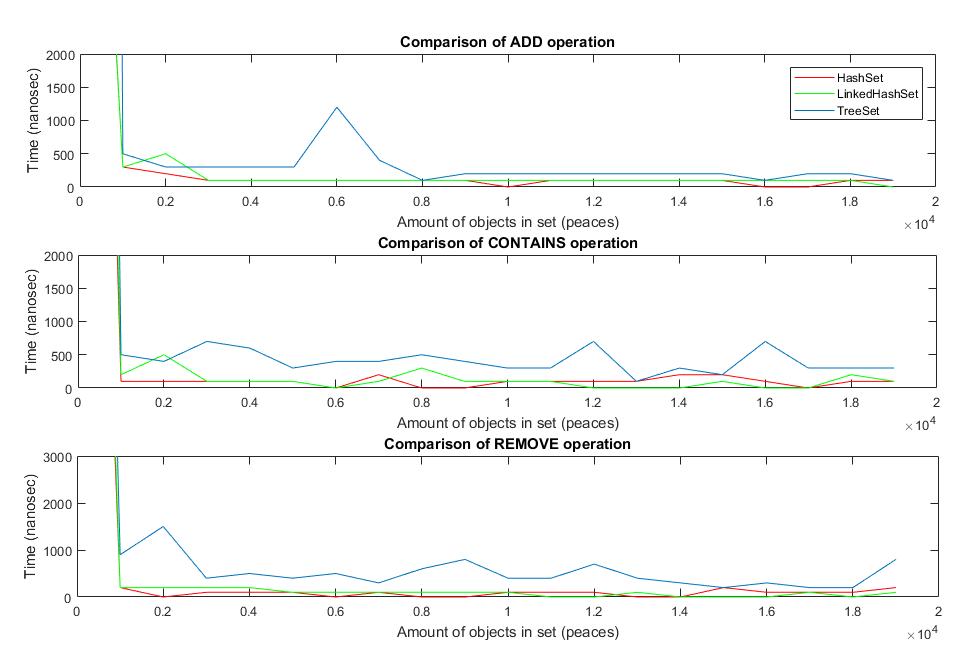
Операцию add выполняю следующим образом: первую половину элементов (10000 элементов) записываю последовательно в конец и получаю в итоге производительность 0(1). После этого начинаю записывать постоянно в индекс, равный половине размера коллекции (в моем случае это 10000). После этого получаю рост затраченного времени для LinkedList коллекции на запись. Связываю это с тем, что теперь для поиска нужного индекса операции add(index,element) приходится проходить всё больше элементов от конца списка. В случае ArrayList получаю лишь небольшое увеличение затраченного времени, связанное с тем что создается новый массив большего размера и туда перекопируется текущий. Разницу работы между LinkedList и ArrayList связываю с тем, что не использовал ListIterator для быстрого нахождения нужного места вставки у LinkedList, а также с хорошей реализованностью и быстрым перекопированием коллекции у ArrayList, ведь элемент по индексу он находит за 0(1).

Что касается операции REMOVE, то каждый раз я удалял из коллекции элемент, который лежал в середине коллекции. Известно, что LinkedList имеет сложность O(1) на удаление (как и на вставку). График это подтверждает (примерное равное время). В свою очередь, ArrayList требует больше времени на операцию remove, что также видно на графике.

Операция GET. Известно, что ArrayList имеет сложность O(1) на поиск, а LinkedList O(n).

График подтвержает, что ArrayList имеет преимущество при поиске элемента.

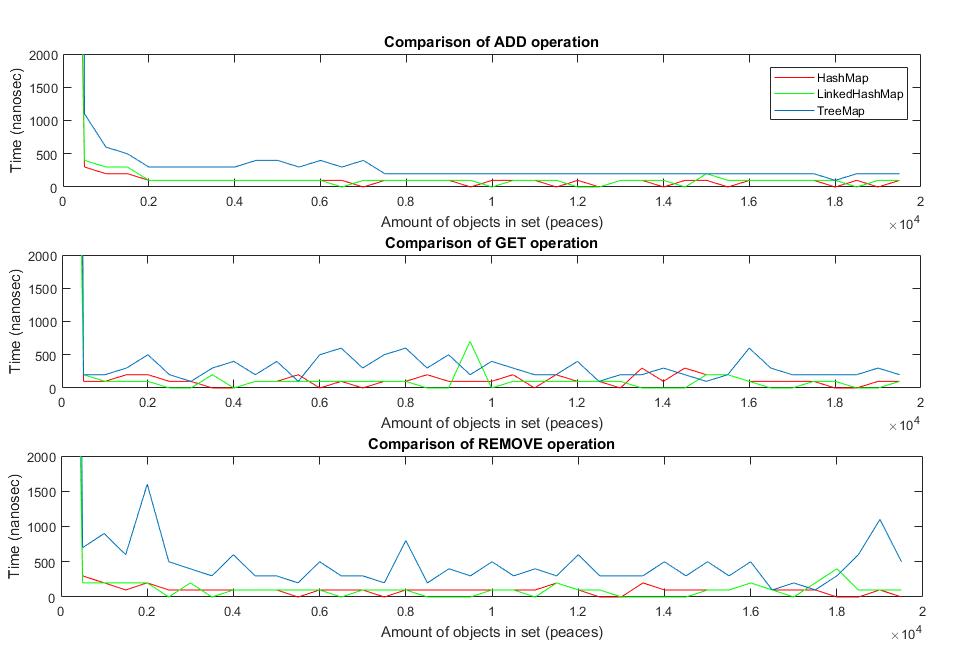
1. Сравнение HashSet, LinkedHashSet и TreeSet.



Также производилась работа с 20000 элементов.

В первую очередь по графикам видно, что независимо от операции, реализация TreeSet работает медленнее других двух реализаций. Это связано с тем, что она сортирует элементы коллекции и выстраивает красно-черное дерево. Сложность всех операция O(log n), что достаточно быстро, но медленне, чем O(1), которой обладают реализации HashSet и LinkedHasSet. Учитывая тот факт, что измерение ведется в наносекундах, эти реализации, согласно графику, имеют схожее время выполнения для каждой из операций.

1. Сравнение HashMap, LinkedHashMap и TreeMap.



Из графиков видно, что, опять же, TreeMap работает медленне, т.к. сортирует элементы коллекции и основана на красно-черном дереве. LinkedHashMap и HashMap затрачивают меньшее время для операций add, get, remove, чем TreeMap. HashMap и LinkedHashMap имеют схожее время выполнения для каждой из операций.